Jurnal SIMETRIS, Vol 6 No 2 November 2015
ISSN: 2252-4983

METODE *NON- HEURISTIC* UNTUK DETEKSI *REFACTORING NON-SOURCE CODE* (*SYSTEMATIC LITERATURE REVIEW*)

Ratih Nindyasari
Fakultas Teknik, Program Studi Teknik Informatika
Universitas Muria Kudus
Email: ratih2502@gmail.com

ABSTRAK

Refactoring perangat lunak merupakan sebuah teknik untuk mentransformasi struktur internal perangkat lunak tanpa merubah fungsionalitas perangkat lunak itu sendiri. *Refactoring* termasuk dalam bentuk evolusi perangkat lunak, dimana hal ini bertujuan untuk meningkatkan kualitas perangkat lunak itu sendiri setelah perangkat lunak mengalami perbaikan, modifikasi, penambahan dan aktivitas perubahan lainnya dalam masa hidupnya. Tujuan penulisan pada *paper* kali ini adalah untuk melakukan *Systematic Literature Review* (SLR) tentang persoalan deteksi refactoring pada level non-source code yang tergolong pada metode non-heuristic. Dengan dilakukan SLR ini terdapat beberapa teknik seperti clustering, searching, Information Retrieval dan analisis regresi dapat digunakan untuk mendeteksi persoalan refactoring pada level non-source code. Dari beberapa teknik-teknik ini masing-masing memiliki kelebihan dan kekurangan. Teknik yang memiliki tingkat komputasi yang tinggi seperti clustering, searching, dan analisis regresi biasanya mampu menghasilkan refactoring yang lebih optimal dan menyeluruh dibandingkan dengan teknik lainnya seperti NLP dan teknik similaritas.

Kata kunci: *refactoring, non-source code refactoring, non-heuristic, systematic literature review.*

ABSTRACT

Software refactoring is one of technique for tranforming software internal structure without change functionality of software. Refactoring include in scope software evolution, it have to increase software quality after reparation process, modification, extend and other change activity of his lifetime. In this paper will do Systematic Literature Review (SLR) for detection of refactoring for non-source code using non-heuricticmethod. There are several techniques or algorithms that can be classified as non-heuristic method such as clustering, searching, information retrieval, and regresion analysis. There are can detection refactoring problem for level non-source code. From some of these, each have advantages and disadvantages. Techniques that have high level of computing such as clustering, searching, and regression analysis commonly capable to produce refactoring result more optimal and comprehensive compared to other techniques such as NLP and similarity techniques.

Keywords: *refactoring, non-source code refactoring, non-heuristic, systematic literature review.*

1. PENDAHULUAN

Refactoring perangat lunak merupakan sebuah teknik untuk mentransformasi struktur internal perangkat lunak tanpa merubah fungsionalitas perangkat lunak itu sendiri [1]. *Refactoring* termasuk dalam bentuk evolusi perangkat lunak, dimana hal ini bertujuan untuk meningkatkan kualitas perangkat lunak itu sendiri setelah perangkat lunak mengalami perbaikan, modifikasi, penambahan dan aktivitas perubahan lainnya dalam masa hidupnya. Selain untuk meningkatkan kualitas, teknik *refactoring* identik dengan proses restrukturisasi perangkat lunak yang bertujuan untuk meminimalkan *error*, misalnya dengan menghilangkan kode yang terduplikasi atau *bad code smell*. Teknik *refactoring* juga berhubungan dengan usaha untuk memperbaiki struktur serta dokumentasi agar mudah dipahami dan dimengerti, sehingga pengembang dapat dengan mudah melakukan pengembangan perangkat lunak pada proses evolusi selanjutnya.

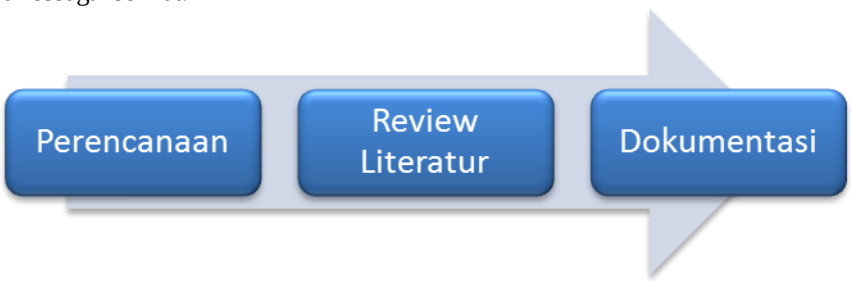
Sejak pertama kali diperkenalkan, proses *refactoring* yang umum dilakukan adalah proses restrukturisasi *source code* dan terdapat juga publikasi yang membahas masalah ini lebih dalam. Pada umumnya, publikasi yang dilakukan peneliti mengacu pada katalog *refactoring* Fowler, dimana pengembangan yang dilakukan berfokus pada salah satu atau sebagian metode implementasi atau deteksi *refactoring* yang diterapkan pada kondisi tertentu. Selain pengembangan metode *refactoring* pada *source code*, terdapat pula penelitian yang membahas masalah *refactoring* yang dilakukan pada level abstraksi yang lebih tinggi dari *source code*, yaitu pada model *high-level*, *refactoring* berdasarkan design *pattern*, dan arsitektur *software* [2].

Beberapa penelitian tentang *refactoring* di level *non-source code* masih memberikan banyak kesempatan untuk proses refactoring di luar lingkup yang telah didefinisikan oleh Fowler. Pada sistematik *Literature Review* (SLR) ini, akan melakukan *review* literatur secara sistematik yang membahas masalah algoritma-algoritma untuk

mendeteksi *refactoring* pada level *non-source code* yang diklasifikasikan dalam metode *non-heuristic* untuk mengetahui sejauh mana riset yang telah dilakukan dalam topik ini. Publikasi yang digunakan sebagai sumber referensi adalah publikasi ilmiah yang diterbitkan pada tahun 2010-2015 dan terdapat pada satu sumber yaitu *google scholar*. Setelah ditemukan semua studi terkait, analisa akan dilakukan untuk mendapatkan jawaban berupa rumusan masalah. Pada akhir tulisan, akan dijelaskan juga hasil kesimpulan dan arahan pengembangan riset lebih jauh untuk jenis topik riset yang sama.

2. METODOLOGI PENELITIAN

Pada bagian ini akan dijelaskan bagaimana langkah-langkah yang harus diikuti untuk melakukan SLR. Langkah-langkah yang dilakukan dalam SLR ini mengadopsi panduan SLR [3] dan [4] yang dilakukan oleh Kitchenham dan Charters serta S. Rochimah, dkk. Pokok utama dalam penelitian ini adalah untuk memberikan informasi mengenai perkembangan penelitian tentang teknik *refactoring level non-source code* dengan metode *non-heuristic*, metode atau algoritma yang diterapkan dalam penelitian *refactoring* , serta kelebihan dan kekurangan dari masing-masing metode. Langkah-langkah atau alur yang harus diikuti dalam penelitian ini ditunjukkan pada gambar 1, adalah sebagai berikut:



Gambar 1. Metodologi Penelitian

2.1 Proses Perencanaan

Pada proses perencanaan ini dilakukan dua sub proses yaitu perumusan batasan melalui *research question* dan metode SLR yang akan dilakukan. Berikut ini adalah penjelasan secara detail dari proses perencanaan, adalah sebagai berikut:

a. Perumusan batasan melalui *research question*
   1) Pendefinisian *refactoring non-source code*, yang merupakan proses untuk mendefinisikan apa yang dimaksud dengan teknik *refactoring* level *non-source code*
   2) Pendefinisian batasan refactoring *non-source code*, merupakan proses untuk mendefinisikan lebih lanjut tentang *refactoring non-source code* serta batasan yang akan dilakukan dalam penelitian
   3) Pendefinisian kata kunci, tahun publikasi, jenis atau tipe literatur yang akan diambil dalam pencarian.

b. Perumusan metode yang akan dilakukan adalah merumuskan langkah - langkah pengerjaan yang akan dilakukan, serta menuliskan kriteria hasil yang diharapkan dari masing-masing tahapan.

2.1.1 Rumusan Batasan (Research Question)

Dalam *paper* kali ini terdapat beberapa *Research Question* (RQ) yang digunakan untuk mengidentifikasi, menganalisa dan me-*review* hasil temuan tentang *refactoring* pada *level non-source code* dengan metode *non-heuristic*. RQ bertujuan untuk mengetahui perkembangan dari penelitian dalam proses *refactoring non-source code* yang telah dilakukan hingga sampai saat ini.

a. RQ1: ada berapa teknik-teknik atau algoritma apa saja yang tergolong dalam klasifikasi metode *non heuristic* digunakan untuk mendeteksi persoalan *refactoring*?
b. RQ2 : Apa saja kelebihan dan kekurangan dari masing-masing Algoritma yang tergolong pada metode *non heuristic*?

2.2 Proses Review

Setelah melakukan proses perencanaan, langkah selanjutnya adalah melakukan proses *review*. Dalam proses *review*, ada beberapa tahapan yang dilakukan, antara lain adalah sebagai berikut:

a. Identifikasi hasil pencarian yang relevan

1) Melakukan pencarian berdasarkan kata kunci yang telah didefinisikan
2) Melakukan filter berdasarkan tahun, jenis atau tipe literatur
3) Melakukan proses pencatatan hasil pencarian pada tahap identifikasi

b. Penyeleksian topik utama penelitian, adalah proses untuk melakukan filter literatur berdasarkan rumusan topik yang telah dilakukan sebelumnya tentang *refactoring non-source code* dengan menggunakan metode *non-heuristic*.
c. Pengukuran kualitas literatur yang ditemukan.
   1) Melakukan identifikasi kualitas dari banyaknya rujukan literatur lain terhadap literatur tersebut
   2) Melakukan proses identifikasi kualitas berdasarkan literatur lain yang ditulis penulis
d. Mengambil data yang dibutuhkan, melakukan ektraksi dari literatur yang diambil untuk dianalisa sebagai bagian pembahasan *research question.*
e. Menganalisa data, adalah proses analisa data untuk menjawab pertanyaan pada rumusan masalah dan menemukan kesimpulan.

2.3 Dokumentasi

Tahapan akhir dalam penelitian ini adalah dokumentasi. Tahap dokumentasi terdiri dari proses menulis hasil laporan dan memvalidasi laporan. Masing-masing proses tersebut dapat dijabarkan sebagai berikut ini:

a. Menulis hasil laporan
   Merupakan proses untuk menulis semua hasil temuan, identifikasi, dan analisa yang telah dilakukan ke dalam sebuah dokumen dengan format tertentu. Sehingga dapat dibaca alur metode yang telah dilakukan secara teratur dan sistematis
b. Melakukan validasi laporan
   Merupakan proses untuk melakukan pengecekan kesesuaian tulisan dokumentasi terhadap penelitian yang telah dilakukan serta melakukan validasi laporan yang sudah benar.

3. HASIL DAN PEMBAHASAN

Pada bagian hasil dan pembahasan ini akan dijelaskan tentang hasil SLR yang telah dilakukan. Hasil SLR yang telah dilakukan dimulai dari proses perencanaan sampai pada tahap proses *review*. Hasil yang dilakukan pada proses perencanaan dapat ditunjukkan dengan jawaban dari RQ. Berikut ini adalah penjelasan dari RQ yang telah dituliskan pada bagian sebelumnya.

3.1 RQ1 : Teknik Atau Algoritma Metode - Metode Non-Heuristic

Berdasarkan pada hasil studi ada sepuluh literatur yang ditemukan dan dianalisa. Terdapat beberapa metode yang digunakan untuk *refactoring*. Metode *clustering* digunakan pada tiga penelitian, yaitu *refactoring level package* untuk menyeimbangkan antara *cohesion* dan *coupling* dengan algoritma *Adaptif K-Nearest Neighbour* (A-KNN)[5], *refactoring* pada *requirement* dalam format *eXtensible Markup Language* (XML) [6], dan refactoring pada komponen arsitektur untuk mencari komponen apa saja yang mirip dengan *clustering* dan selanjutnya dilakukan proses restrukturisasi [7]. Penelitian [8] menggunakan metode pendeteksian *refactoring* pada *Refined Process Tree* dari repositori process model dengan mencari kemiripan *string label* dan kemiripan fragmen. Metode *Natural Language Processing* (NLP) digunakan dalam penelitian [9] untuk melakukan *refactoring activity label* pada process model. Model Markov dan algoritma apriori digunakan pada proses *refactoring* berdasarkan *aspect oriented programming* [10]. Dua penelitian menggunakan metode *searching* untuk memudahkan *requirement traceability* [11] dan mencari model design perangkat lunak yang baru berdasarkan *input design quality model* [12]. Algoritma *univariate logistic refression analysis* digunakan pada *Software Quality Metrics* untuk menganalisa kesempatan *refactoring* dan mendapatkan rekomendasi bagian perangkat lunak yang akan direstrukturisasi [13]. Penemuan terakhir adalah algoritma *Max Flow Min Cut* digunakan untuk mencari *bad smell* antar *class* [14].

Setelah melakukan klasifikasi literatur berdasarkan pada metode *non-heuristic* yang digunakan untuk mendeteksi persoalan *refactoring*. Algoritma-algoritma yang terklasifikasi dalam metode *non-heuristic* dapat dilihat pada Tabel 1.

3.2 RQ2 : Kelebihan dan Kekurangan Algoritma - Algoritma Dari Metode Non-Heuristic

Kelebihan dari pengaplikasian metode *non heuristic* untuk mendeteksian *refactoring* yang sifatnya umum dari berbagai pendekatan adalah teknik *clustering* [5][6][7], *searching* atau pencarian [11][12] dan

[8][9][10][13][14]. Dengan menggunakan penerapan dari metode *non-heuristic* ini hasil deteksi yang diperoleh akan lebih optimal dibandingkan dengan metode *heuristic*.

Akan tetapi kekurangan dari metode *non-heuristic* adalah meskipun pendeteksian dilakukan secara menyeluruh, sebuah algoritma cenderung hanya mampu mendeteksi kesempatan *refactoring* dengan karakteristik yang sama. Sehingga sebuah algoritma belum tentu dapat mendeteksi semua kesempatan *refactoring*. Misalnya pendeteksian *refactoring* dengan metode pendeteksian kemiripan fragmen pada *Refined Process Structure Tree* hanya dapat mendeteksi 4 kesempatan *refactoring* pada process model dari 7 katalog *refactoring* yang ada [8]. Demikian pula implementasi metode NLP yang hanya mampu mendeteksi *activity label* dalam bahasa Inggris dan masih memiliki masalah dalam hal kalimat majemuk, kata tidak beraturan, dan masalah bahasa lainnya [9]. Dengan kata lain, lingkup atau jangkauan pendeteksian *refactoring* dari sebuah algoritma tertentu bersifat terbatas, dimana keterbatasan ini bergantung pada kemampuan masing-masing algoritma tersebut. Tabel 2. Berisi tentang penjelasan dari kelebihan dari masing-masing algoritma dari metode *non-heuristic*.

Berdasarkan pada hasil *review* literatur tentang teknik-teknik atau algoritma yang dapat digunakan untuk mendeteksi refactoring pada level *non-source code* ini, tiap-tiap algoritma memiliki kelebihan dan kekurangan. Teknik *clustering* mampu menghasilkan deteksi refactoring lebih optimal dibandingkan dengan metode deteksi similaritas. Namun teknik *clustering* pada umumnya lebih rumit dan membutuhkan waktu komputasi yang tinggi. Begitu juga dengan teknik *searching* dan *IR*.

Tabel 1. Daftar SLR dan algoritma yang terklasifikasi dalam metode *non-heuristic*

Judul Literatur	***Algoritma***	***Deskripsi***
Identifying refactoring opportunities in process model repositories	Teknik Similaritas *string label* dan fragmen	Pada literatur ini digunakan teknik similaritas *string label* dan fragmen yang dilakukan pada *Refined Process Tree* dari repositori model.
On the refactoring of activity labels in business process models	Metode NLP (*Natural Language Processing*)	Pada literatur ini menggunakan metode NLP untuk melakukan *refactoring activity label* pada model proses.
Software refactoring at the package level using clustering techniques	Teknik *clustering* dengan A-KNN	Pada literatur ini menggunakan metode *clustering* yaitu A-KNN dengan melakukan *refactoring* pada *level* paket.
Toward automated refactoring of crosscutting concerns into aspects	Algoritma Apriori	Mendeteksi *refactoring* berdasarkan pada *aspect oriented programming*
Supporting Requirements Traceability through Refactoring	Teknik *searching*	Mendeteksi *refactoring* yang bersumber dari hubungan antar artefak dari suatu perangkat lunak
Traceability-Enabled Refactoring for Managing Just-In-Time Requirements	Algoritma *Information retrieval* dan Teknik *Clustering*	Mendekteksi *refactoring* berdasarkan informasi *bug* atau *error* maupun *bad smell* sehingga didapatkan kebutuhan perangkat lunak baru
Search-based Refactoring for software maintenance	Algoritma *searching*	Mendeteksi *refactoring* yang dilakukan pada matriks kualitas perangkat lunak
Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics	Teknik analisis regresi logistik	Mendeteksi *refactoring* pada *class* sehingga dapat dideteksi *extract subclass refactoring* (ESR)
A technique for automatic component extraction from object-oriented programs by refactoring	Teknik *clustering*	Melakukan *refactoring extract class* dari relasi antar *class* yang dimodelkan dengan *graph*
Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures	Algoritma *MaxFlowMinCut*	Melakukan *refactoring* dari *graph* yang berisi hubungan antar *class*, yaitu kohesi dan *coupling*

Tabel 2. Kelebihan dan kelemahan algoritma metode *non-heuristic*

Algoritma/Teknik	*Kelebihan*	*Kelemahan*
Clustering	Hasil deteksi lebih optimal, dapat mendeteksi *refactoring* secara menyeluruh	Kompleksitas tinggi
Searching	Hasil deteksi lebih optimal, dapat mendeteksi *refactoring* secara menyeluruh	Kompleksitas tinggi
Deteksi Similaritas	Ringan, tidak memerlukan waktu komputasi yang lama.	Hanya dapat mendeteksi *refactoring* sebagian/tidak menyeluruh
NLP	Hasil deteksi stabil, optimal	Memiliki keterbatasan ketika mendeteksi *aktivity label* yaitu belum bisa *multilanguage*, hanya bisa mendeteksi jika menggunakan bahasa inggris.
univariate logistic regression analysis	Menghasilkan kualitas matriks yang tinggi	Kompleksitas tinggi, komputasi tinggi
Max Flow Min Cut	Hasil deteksi	-

4. KESIMPULAN

Pada *review literature* kali ini menghasilkan kesimpulan bahwa *refactoring level non-source code* dengan metode *non-heuristik* dapat dilalui dengan beberapa teknik atau algoritma diantaranya adalah : *clustering*, *searching*, similaritas, NLP, analisa regresi, dan algoritma *Max Flow Min Cut*. Masing-masing teknik atau algoritma memiliki kelebihan dan kekurangan masing-masing. Untuk teknik *clustering*, *searching*, dan analisa regresi dalam beberapa literatur mampu menghasilkan hasil deteksi *refactoring* yang menyeluruh dan optimal. Namun komputasinya tinggi akibat adanya kompleksitas yang tinggi. Sedangkan teknik lainnya seperti deteksi kemiripan atau similaritas, NLP, dari beberapa literatur mengatakan bahwa hasil deteksi kurang menyeluruh.

DAFTAR PUSTAKA

[1] Martin Fowler, *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Weasley Longman Publishing Co. Inc., 1999.

[2] Refactoring Browser. [Online]. http://c2.com/cgi/wiki?RefactoringBrowser

[3] Barbara Kitchenham and Stuart Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Keele University, UK, Technical Report EBSE 2007.

[4] S.Rochimah, S. Arifiani and V.F. Insanittaqwa, "Non-Source Code Refactoring : A Systematic Literature Review, " International Journal of Software Engineering and Its Applications, vol.9, No.6, 2015

[5] A. Alkhalid, M. Alshayeb, and S.A. Mahmoud, "Software refactoring at the package level using clustering techniques," *IET Software*, vol. 5, no. 3, pp. 274-286, 2011.

[6] Nan Niu, Tanmay Bhowmik, Hui Liu, and Zhendong Niu, "Traceability-Enabled Refactoring for Managing Just-In-Time Requirements," in *22nd IEEE International Conference on Requirements Engineering (RE)*, 2014, pp. 133-142.

[7] Hironori Washizakia and Yoshiaki Fukazawab, "A technique for automatic component extraction from object-oriented programs by refactoring," *Science of Computer Programming*, vol. 56, pp. 99-116, 2005.

[8] Remco Dijkman, Beat Gfellar, Jochen Kuster, and Hagen Volzer, "Identifying refactoring opportunities in process model repositories," *Information and Software Technology*, vol. 53, pp. 937-948, 2011.

[9] Henrik Leopold, Sergey Smirnov, and Jan Mendling, "On the refactoring of activity labels in business process models," Informations Systems, vol. 37, pp. 443-459, 2012.

[10] Santiago A. Vidala and Claudia A. Marcosa, "Toward automated refactoring of crosscutting concerns into aspects," The Journal of Systems and Software, vol. 86, pp. 1482-1497, 2013.

[11] Anas Mahmoud and Nan Niu, "Supporting Requirements Traceability through Refactoring," in 21st IEEE International Conference of Requirements Engineering (RE), 2013, pp. 32-41.

[12] Mark O'Keeffe and M.O. Cinnéide, "Search-based Refactoring for software maintenance," The Journal of Systems and Software, vol. 81, pp. 502-516, 2008.

[13] Jehad Al Dallal, "Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics," Information and Software Technology, vol. 54, pp. 1125-1141, 2012.

[14] Gabriele Bavota, Andrea De Lucia, and Rocco Oliveto, "Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures," The Journal of Systems and Software, vol. 84, pp. 397-414, 2011.